DUPLICATION OF MASTER DISKETTE PROCEDURE SUMMARY

Your first disk operation should be to write-protect and duplicate your Master Diskette. The following steps describe the necessary procedure for duplicating the diskette.

1. Turn on disk drive. Wait for BUSY light to go out.
2. Remove Master Diskette from white, protective envelope.
3. Place write-protect tab over notch on Master Diskette.
4. Insert Master Diskette into Disk Drive #1 and close drive door.
5. Turn on computer console. DOS will "boot" into RAM.
6. Type DOS [RETURN], if cartridge is inserted.
7. Remove Master Diskette and insert a blank diskette or one you wish to erase.
8. After the DOS Menu and SELECT ITEM prompt appear, Type I [RETURN] to format diskette.
9. Type 1 [RETURN] in response to WHICH DRIVE TO FORMAT? prompt message.
10. Type Y [RETURN] in response to TYPE "Y" TO FORMAT DRIVE 1 prompt message.
11. When SELECT ITEM prompt message appears, type H [RETURN] .
12. Type Y [RETURN] in response to TYPE "Y" TO WRITE NEW DOS FILE? prompt message.
13. Prompt message WRITING NEW DOS.SYS FILE displays on the screen.
14. When SELECT ITEM prompt message appears, the duplication of the diskette is complete.

TABLE OF CONTENTS

This ATARI(R) Disk Operating System (DOS) Reference Manual assumes that the user is familiar with ATARI BASIC. It explains the commands and statements used by the Disk Operating System (initial release-9/24/79) to move data to and from the ATARI 810[TM] Disk Drive(s).

The first section explains the procedure for powering-up the console and powering up and initializing the Disk Drive(s). It also defines the notations and conventions used throughout the manual.

The second section describes the ATARI 810[TM] Disk Drive and a little about its operation. From this, the manual proceeds to describe the diskettes and how they are organized to accept data. Section 3 also contains a "trouble-shooting" section on BOOT ERRORS and possible reasons for their occurrence.

Section 4 describes the interaction that takes place within the Disk Operating System itself when it is in operation. The two main files within the DOS are described in terms of their relationship to the DOS Menu. This section also explains the parameters and "wild cards" recognized by DOS.

Sections 5 and 6 contain descriptions of the DOS Menu selections and the BASIC commands used with disk operations. Each of these provides an example of its use.

The appendices give useful information including the memory map, a glossary, error codes, and hints on how to obtain more useable RAM.

SECTION 1. GENERAL INFORMATION

This section reviews the procedure for powering-up and initializing an ATARI Personal Computer System with at least one ATARI Disk Drive attached. It also defines the notation conventions and general information that is used throughout this manual. It does not contain information regarding the attachment of disk drive(s) to the computer console. That information is contained in the ATARI Disk Drive Operator's Manual.

DISK DRIVE POWER-UP AND INITIALIZATION

After you have checked the connections and placement of your disk drive(s), use the following procedure to power-up your system and to initialize the disk drive(s). This initialization procedure is also called a "bootstrap" operation or "autoboot."

1. Turn on television set.
2. Turn on Disk Drive unit(s). The BUSY light(s) will come on and will stay on until each drive unit is initialized.
3. Turn on any other peripheral devices; e.g., printer.
4. Insert DOS diskette in Disk Drive #1 and close disk drive door.

   NOTE: The Master Diskette DOS should always be placed in Disk Drive #1 (see Drive Code Settings, Section 2).

5. Turn on computer console.
6. If you get a BOOT ERROR, turn off computer console for approximately 5 seconds, then turn it on again. If the BOOT ERROR message persists, check all connections and make sure the the drive door is closed. If everything seems to be alright, check the section entitled BOOT ERRORS on page 21.

DEVICE IDENTIFICATION

Each ATARI device, including the disk drives, has an identification letter that allows you to access it. These identification codes are given below with a short description of each device:

C: ATARI 410[TM] Program Recorder. This is both an input and output device. If you want to save a program on tape in its tokenized form, use either the CSAVE or SAVE "C:" command.

D[1,2,3,4]: ATARI 810[TM] Disk Drives. The disk drives are both input and output devices. To save a program on diskette, select the drive you want to use; e.g., Disk Drive #2, and use the command SAVE "D2:PROG1.BAS." Device Identification D: is equivalent to D1:.

E: Screen Editor. This input/output device uses the keyboard and display (see K: and S:) to simulate a screen editing terminal. Writing (output) to this device causes the data to appear on the display starting at the current cursor position. Reading (input) from this device activates the screen editing process and allows you to enter and edit data.

K: Keyboard. This input only device allows you to read the converted (ATASCII) keyboard data as you press each key.

P: Line Printer. This output only device prints ATASCII characters, usually a line at a time. This device identification is used for the ATARI 820[TM] Printer, the ATARI 822[TM] Thermal Printer, and the ATARI 825[TM] 80-column Printer.

R[1,2,3,4]: ATARI 850[TM] Interface Module. This device handles both input from and output to RS232C-compatible peripheral devices (and output only to a printer requiring an 8-bit parallel port accessed through P:).

S: TV Monitor. This input/output device allows the user to read characters from and write characters to the display using the cursor as the screen addressing mechanism.

Throughout this manual, you will see these device identifications used in both the DOS Menu options and BASIC commands used with DOS.

FILE SPECIFICATIONS

Information is stored on a diskette in files. Files are classified into two types: program files and data files. Data files usually contain data used by a program file. A program file contains the instructions to perform a certain task.

When referring to a file on a diskette, use a file specification (or filespec). A filespec consists of up to six elements. Figure 1-1 illustrates the six possible elements of a filespec. In BASIC, quotation marks are required when accessing a file.

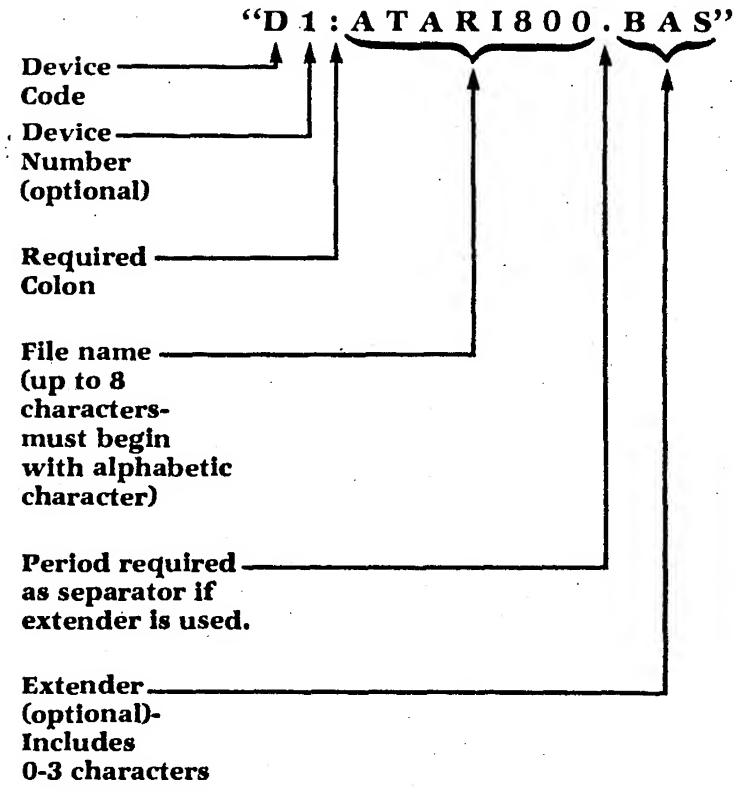

Figure 1-1 EXAMPLE OF A FILESPEC

Filenames

Filenames follow specific rules. If a filename does not follow these rules, you will see an ERROR-165 (File Name Error) message on the screen. The rules for filenames are:

1. The maximum length of a filename is 8 characters.
2. The only characters that can be used are A through Z and 0 through 9.
3. The first character is always an alphabetic character.

DOS.SYS is a filename reserved by DOS.

Extenders

A three-character extender can be added to a filename to indicate the type of data in a file.

Currently, you can use any legal combination of letters and numbers for an extender, e.g., .SYS for system files, .BAS for BASIC program

files, .DAT for data files, .LST for list files, .OBJ for binary files, .SRC for source files, .MUS for Music Composer files, etc. You can use up to 3 legal characters for extenders. Characters beyond 3 characters are truncated and ignored.

Examples of Filenames

RATATAT	Legal name.
ATARI.BAS	Legal name.
3ATARI.DAT	Illegal name. First character is not an alphabetic letter.
ATARI22.XYZ	Legal name.
ATARI#	Illegal name. # character cannot be used.
A1234567.829	Legal name.
B ATARI.BAS	Illegal name. No spaces allowed.
ATARI.BASIC	Legal name. (DOS will truncate the last 2 letters of the extender.)
DOS.SYS	Illegal name. Reserved for DOS file.
DOSSYS	Legal name.

DOS Options Used With Filenames

Certain letters can be added to a filename to perform a specific task.

/A means to append data to an existing file on diskette. In the course of using the SAVE BINARY FILE selection on the DOS Menu, entering D:BINFILE/A,5000,52FF would append the contents of locations 5000 through 52FF to BINFILE, which has already been stored on disk drive #1.

/N means no verification of an operation. In using the DELETE FILE selection on the DOS Menu, entering D2:DELTA.BAS/N bypasses the normal verification prompt message.

[RETURN] Press the [RETURN] key on the keyboard.

[ ] Brackets. Brackets enclose optional items.

... Ellipsis. An ellipsis following an item in brackets indicates that you can repeat the optional item any number of times, but are not required to do so.

{ } Items stacked vertically in braces indicate you have a choice as to which item you want to insert. Select only one in your statement or command.

CAPITAL LETTERS Capital letters are used to indicate commands, statements, and other functions that you must type exactly as they appear.

, . / : ; " These punctuation marks must be typed as shown in the format of a command or statement. However, do not type brackets or braces.

adata ATASCII data. Any string of ATASCII characters, excluding commas and carriage returns. Refer to the ATARI BASIC Reference Manual, Appendix C.

aop Arithmetic Operator.

lop Logical Operator.

cmdno Command Number (used in XIO commands).

exp Any Expression. In this manual, expressions are divided into three types: arithmetic expressions, logical expressions, and string expressions.

aexp Arithmetic Expression. Generally composed of a variable, function, constant, or two arithmetic expressions separated by an arithmetic operator (aop).

aexp1 Arithmetic Expression 1. This arithmetic expression represents the first auxiliary I/O control byte when used in commands such as OPEN.

aexp2 Arithmetic Expression 2. This arithmetic expression represents the second auxiliary I/O control byte when used in commands such as OPEN. Usually it is set to 0. If, however, you want to direct the ATARI 820[TM] Printer to sideways printing, you would set this arithmetic expression to 83.

lexp Logical Expression. Generally composed of two arithmetic or two string expressions separated by a logical operator. Such

an expression evaluates to either a 1 (logical true) or a 0 (logical false).

sexp String Expression. Can consist of a string variable, string literal (constant), or a function that returns a string value.

filespec File Specification. A string expression that refers to a device such as the keyboard or to a disk file.

iocb Input/Output Control Block (IOCB). An arithmetic expression that evaluates to a number from 1 to 7. The IOCB is used to refer to a device or file. IOCB 0 is reserved for BASIC and cannot be used.

lineno Line Number. A constant that identifies a particular program line in a deferred mode BASIC program. A line number can be any integer from 0 through 32767. Line numbering determines the order of program execution.

var Variable. Any variable. In this manual, variables are classified as arithmetic variables (avar), matrix variables (mvar), or string variables (svar).

avar Arithmetic Variable. A location where a numeric value is stored. Variable names can be from 1 to 120 alphanumeric characters, but must start with an unreversed, upper case alphabetic character.

mvar Matrix Variable. An element of an array or a matrix. Matrix variables can be subscripted. A matrix variable is a number, variable, or expression placed in parentheses immediately following the name of the array or matrix. For example, A(1), A(ROW), A(X+1,Y-3).

svar String Variable. A location where a string of characters can be stored. The same rules given for the arithmetic variable (avar) apply to the string variable with the additional restriction that the string variable name must end in $. String variables can be subscripted.

VERSIONS AND RELEASES

Currently, the DOS Master Diskette is Version I. Subsequent versions will be designated as DOS II, DOS III. etc. This manual describes only DOS I. As ATARI releases new versions of DOS, new documentation describing those versions will also be released.

SECTION 2. ABOUT THE DISK DRIVE

An ATARI 810[TM] Disk Drive provides fast, reliable data storage. When attached to your ATARI Personal Computer System, you can:

1. Store programs on diskette..
2. Retrieve programs from diskette.
3. Create and add to data files needed by programs.
4. Make copies of disk files.
5. Erase old files from a diskette.
6. Load and save binary files.
7. Move files to and from memory, the screen, diskette, printer, and to and from any new peripherals that ATARI, Inc. will introduce.

The ATARI Disk Drives are "smart." Each contains a microprocessor with its own software in addition to the logic and hardware necessary for the magnetic head to move back and forth "across" a floppy (flexible) diskette.

Your ATARI Personal Computer System with 16K RAM can accommodate up to four ATARI 810 Disk Drives. Each drive operates independently of the others.

ATARI 810 Disk Drive

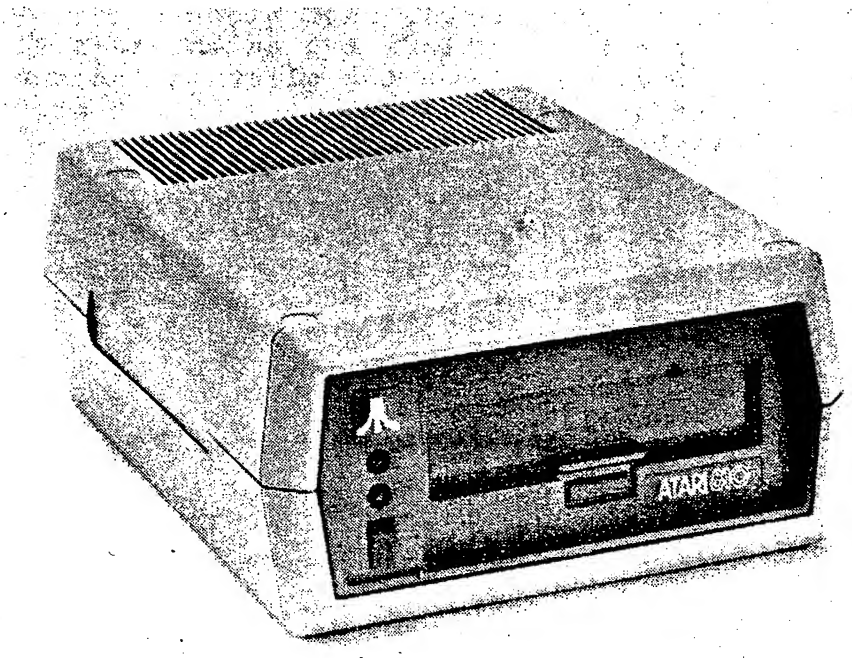

FIGURE 2-1. ATARI 810 DISK DRIVE

The ATARI 810 Disk Drive is a single drive with single density recording capabilities. It uses standard 5 1/4" flexible diskettes, each of which stores 88K (88 thousand) bytes. The 810 Disk Drive contains a built-in microprocessor which gives it an automatic stand-by capability. This means that the Disk Drive motor is not in constant operation, but waits to be "told" when it is needed. The ATARI 810 Disk Drive Operator's Manual describes many of the Disk Drive's features.

MULTIPLE DISK DRIVE NUMBERING

When you attach the Disk Drive(s) to your ATARI Personal Computer System, you should check the back of each drive to ensure that each one is specified to a different DRIVE CODE NUMBER (Figure 2-1).

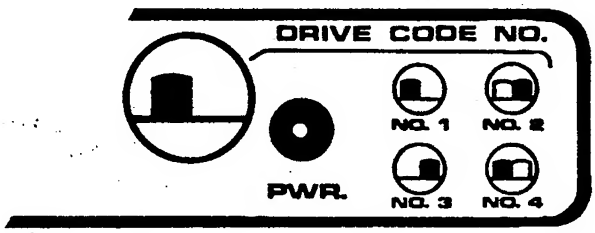

FIGURE 2-2. DRIVE CODE SETTINGS

The disk drive in the above figure would be designated as Drive #1 because the white tab is behind the black tab on the left side. To change the drive to Disk Drive #2, move the black tab to the right, but leave the white tab on the left. If you move both tabs, the disk drive will be set to Disk Drive #3 position.

You can designate the ATARI 810 Disk Drives in any order; e.g., if you have two 810 Disk Drives, either can be Drive #1 and the other can be Drive #2, #3, #4. Remember that DOS will not be able to boot unless the drive in the #1 position contains a Master Diskette or its duplicate.

DISK DRIVE OPERATION

When you have set the drive code positions and connected the drive(s) to your ATARI Personal Computer System, you are ready to insert a diskette and begin disk drive operations.

SECTION 3. DISKETTES

This section describes diskettes, their organization, and care. It also covers the method for write-protecting a diskette so that valuable files are safe from being overwritten or erased.

DISKETTE DESCRIPTION

ATARI Diskettes are thin mylar circular sheets covered with an oxide similar to that used on recording tape. Each ATARI Diskette is about 5 1/4" in diameter and each is sealed in a special, black jacket designed to protect it from being bent, scratched, or contaminated. Figure 3-1 illustrates a diskette.

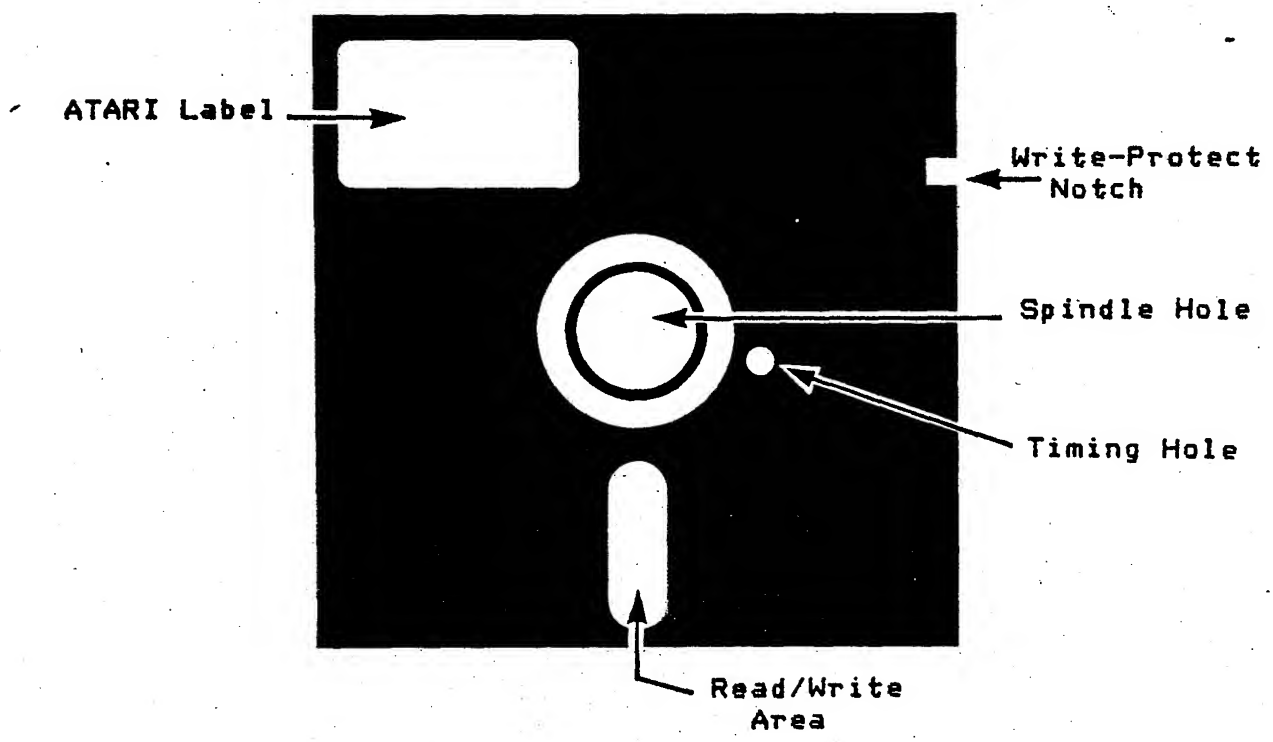

FIGURE 3-1. DISKETTE IN PROTECTIVE COVERING

NOTE: DO NOT ATTEMPT TO REMOVE THE THE DISKETTE FROM ITS BLACK PROTECTIVE COVERING.

The read/write area on each diskette is exposed and the diskette can be damaged if you scratch or fingerprint this area, expose it to bright sunlight, or drop ashes on it. To prevent this from happening,

it is recommended that all diskettes not in use be kept in the white protective envelopes provided with them.

When the diskette is inserted into the Disk Drive, the spindle hole is automatically placed on the drive hub and the diskette is seated. The circular diskette spins within its protective jacket. When you access the diskette, the magnetic head is placed over the read/write area.

DISKETTE WRITE-PROTECT

A sheet of large file identification labels and a sheet of small, adhesive tabs are included in each Master Diskette box(CX8101) and in each box of 5 Blank Diskettes(CX8100). To write-protect a diskette, remove one adhesive tab from the sheet and fold it over the notch on the edge of the diskette (Figure 3-2). It is recommended that you write-protect your Master Diskette immediately and any other diskettes containing valuable files. Write-protecting a diskette prevents it from being inadvertantly overwritten. If you attempt to write on a write-protected diskette, an ERROR-144 displays on the screen.

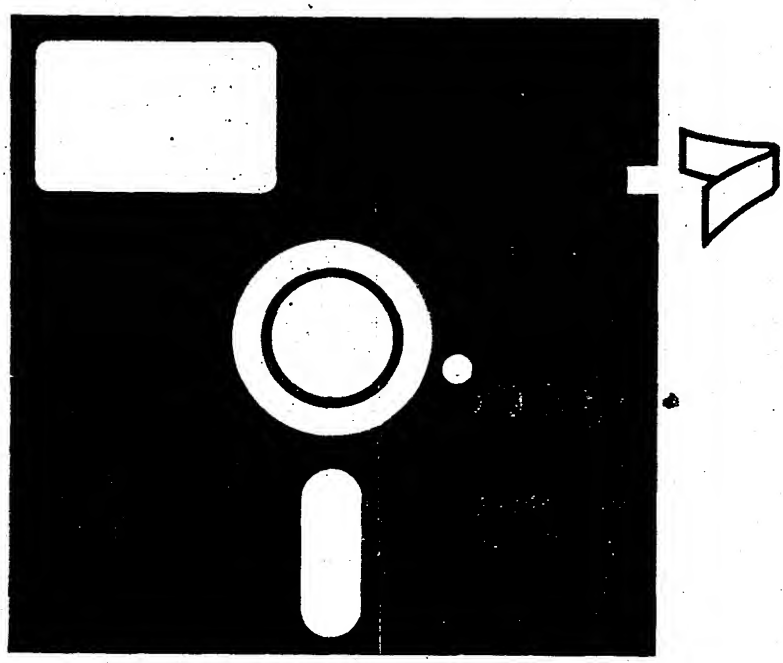

FIGURE 3-2. WRITE-PROTECTING A DISKETTE

DISKETTE ORGANIZATION

Before you can write on a blank diskette, you must "organize" the surface so that the DOS will know where the information is located. This is done by formatting a diskette into tracks and sectors. (Refer to Section 5, I. DISK FORMAT.) After a diskette has been formatted, it is logically divided into 40 tracks with 18 sectors per track. Each of these single density diskettes thus has a total of 720 sectors on which you can write information (see Figure 3-3).

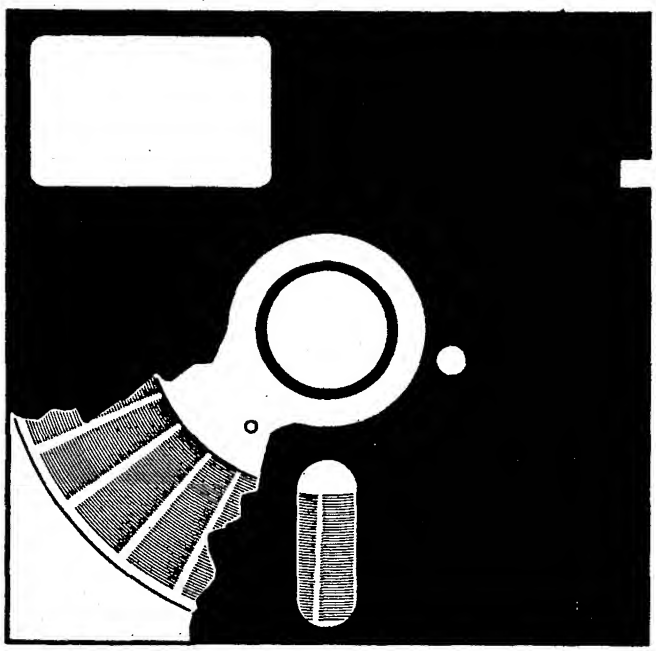

FIGURE 3-3. FORMATTED SINGLE DENSITY DISKETTE

Eleven of the sectors are allocated by the DOS for "special duty" so that they are not available to you.

```
 1 sector used for boot
 8 sectors used for Directory
 1 sector used for Volume Table of Contents
 1 sector (number 720) is not addressable, so is unused.
---
11 TOTAL
```

Each of the remaining 709 sectors can store 128 bytes of information, 3 bytes of which are used for overhead. That gives a total byte

capacity per single density diskette of 88,625 bytes.

If a diskette has the DOS on it, its storage capacity is reduced since the DOS file itself uses approximately 9K bytes. It is not necessary to have the DOS file on every diskette, but you will have to remember to load the DOS from your Master Diskette before inserting your program diskette.

When you store data on a diskette, the disk drive converts the data it receives from the console into coded electrical pulses. These pulses magnetize minute areas of the oxide coating of each diskette while the diskette is spinning.

When you retrieve data from a diskette, the disk drive positions the magnetic head so that the area of the diskette where the data is stored passes beneath it. The disk drive's microprocessor controls the positioning and timing for the diskette.

DISKETTE INSTALLATION

Inserting the diskette into the ATARI 810 Disk Drive is a simple, but very important procedure. If the diskette is improperly inserted, it can cause boot errors. Improper diskette insertion can also cause damage to the diskette itself.

To insert the diskette, remove it from the white, protective envelope and, holding it as shown in Figure 3-4, gently slide it into the drive. Be sure that the notch is on the LEFT side.

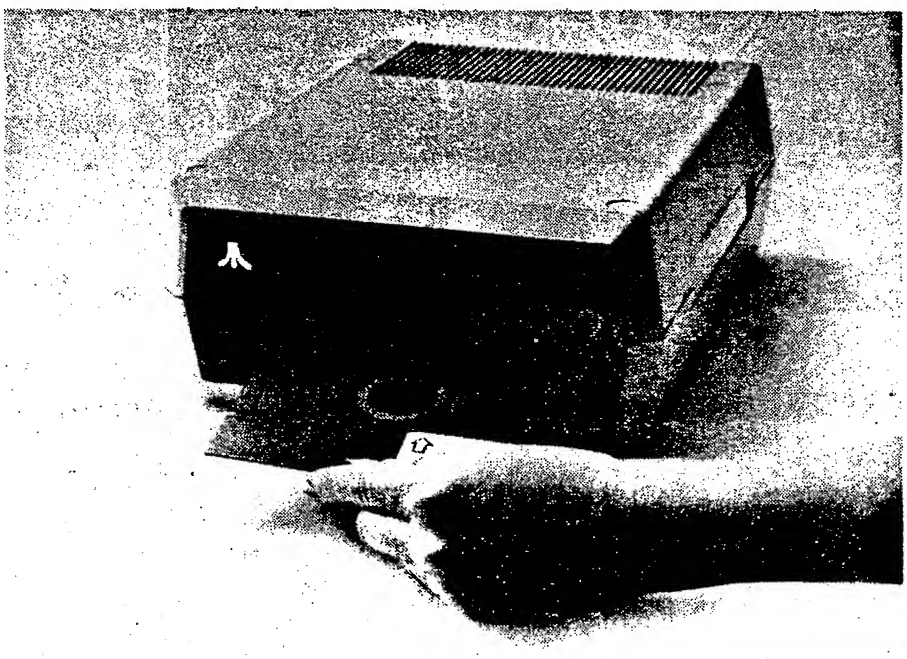

Figure 3-4 DISKETTE INSERTION

DISKETTE STORAGE

Since your diskettes are flexible, they are subject to being damaged. Always keep your diskettes not in use in their white, protective envelopes and store them vertically. It is recommended that you store your diskettes at least 12" to 18" from your television set or any other possible source of magnetic fields. It is also recommended that you keep them away from any source of heat.

If you handle your diskettes with care and store them properly, you will minimize the chance of losing valuable data.

BOOT ERRORS

If you have inserted the diskette and get a BOOT ERROR message displayed over and over again on the screen, turn off the computer console, re-read the procedure and try it again (refer to Section 1). BOOT ERRORS can occur if:

1. The inserted diskette does not have the DOS on it.
2. The disk drive was powered up after the computer console was turned on.
3. The disk drive is not properly connected to the computer console.
4. The transformer plug has loosened in its wall socket.
5. The power plug has loosened in the PWR socket in the disk drive.
6. The diskette was improperly inserted.
7. The diskette has been scratched, warped, or marred. In this case, use another diskette.
8. The drive code setting is not correct.

If you are sure that it is none of these probems, you should use the Master Diskette to boot up, re-insert the problem diskette, and save any accessible files on yet another diskette. Then, reformat the "problem" diskette and try to use it again.

SECTION 4

DISK OPERATING SYSTEM

DOS (pronounced doss) is an acronym for Disk Operating System. It is an extension of the ATARI Operating System (OS) that allows you to interface with a disk drive so that information can be passed between the diskette and the computer memory (RAM).

DOS DESCRIPTION

The ATARI Disk Operating System contains two main parts:

- a Disk Utility Package (DUP)
- a File Management Subsystem (FMS)

This section describes the interaction that takes place between the OS and DOS when the system is powered up and the disk drive is "booted." It also provides a description of the two main parts contained in DOS.

When you power up your disk drive(s) and the computer console, the Operating System executes a bootstrap loader that brings a special file called DOS.SYS into RAM and begins executing the initial code in that file. DOS.SYS contains both the File Management Subsystem and the Disk Utility Package.

When DOS I is loaded into the computer RAM, it occupies a special area in RAM that does not interfere with the memory locations allocated for BASIC or Assembly Language programs. The Memory Map in Appendix D shows the RAM locations occupied by the DOS. After the disk drive system has been booted and the DOS.SYS file is loaded, the ATARI Operating System turns control of the system to the cartridge. If no cartridge has been inserted, OS gives control of the system directly to the Disk Utility Package.

Disk Utility Package

The Disk Utility Package (DUP) programs allow you to display the DOS Menu by calling and executing its DUP Executive program. The Executive program, besides displaying the DOS Menu, takes Menu input and prints the module entry message for each Menu option; e.g., DELETE FILESPEC for Menu selection D, and executes the selected utility. The DUP programs also allow easy access to the File Management Subsystem (FMS) without your having to understand the logical structure of FMS.

File Management Subsystem

The File Management Subsystem gives you a simpler way of storing data on a diskette by putting a logical structure on top of the formatted

diskette. Because of the File Manager program, you don't have to keep track of all the sectors a program occupies, which sectors they are, or how to find a particular file. FMS relieves the user of all that reponsibility.

The FMS also "controls" the operations performed on a file: OPEN, CLOSE, PUT, and GET (see Section 6). In addition, it defines the subfunctions displayed on the DOS Menu that are accessed by DUP. The subfunctions that are not defined by FMS are BINARY LOAD, BINARY SAVE, RUN AT AT ADDRESS, RUN CARTRIDGE, COPY FILE, DUPLICATE FILE and DUPLICATE DISK.

OPERATION WITHOUT INSERTED CARTRIDGE

When no cartridge has been inserted, the OS gives control directly to the DOS Menu and that is what appears on the screen. However, if you have inserted a cartridge, you must type DOS [RETURN] before the DOS Menu will appear on the screen.

PARAMETERS

A parameter is additional information (sometimes optional) that specifies how the command is to operate. If more than one parameter is required, separate them with commas. For example, the BINARY LOAD selection requires a START and END. A parameter can be a filename or a hexadecimal number. If you enter a parameter and decide against it, press [BREAK]. The selected subroutine will not be executed and a SELECT ITEM prompt message will appear on the screen.

WILD CARDS

ATARI DOS recognizes two "wild cards" which can be substituted for characters in a filename. Wild cards are represented by the special characters ? and *.

The ? is used in place of any single valid character. If there are 25 files on a diskette and you want to list to the screen only those five-letter files beginning with PROB and ending with .BAS, you would type PROB?.BAS. (See A. DISK DIRECTORY.) This wild card can only be substituted for a single character. To substitute for a variable number of characters, there is another, more flexible wild card.

The * stands for any valid combination of characters in a filename or an extender. The following examples illustrate the use of the *.

EXAMPLES:

*.BAS	will list all the program files on a diskette in disk drive #1 that end in .BAS.

D2:*.* will list all the program files on the disk drive 2 diskette.

PRO*.BAS will list all the program files on diskette of disk drive #1 that begin with PRO and have .BAS as the extender.

Letters or numbers (but not a period) to the right of the asterisk (*) are ignored. In other words, *T.BAS would appear to the computer as *.BAS and PROT.*S would be the same as PROT.*.

You can load a program from the diskette using a wild card in the file name if there is no more than one file to which it is applicable; e.g., if while the DOS is searching for a file PRO*.BAS and it finds a PRO1.BAS and a PROB.BAS, an ERROR-21 (Load File Error) will appear on the screen.

SECTION 5

DOS MENU SELECTIONS

This section describes each of the selections that appear on the Disk Operating System Menu. The DOS Menu is "self-prompting." When you type the letter of one of the selections; e.g., A, K, D, etc., in response to the SELECT ITEM prompt, the DUP program prints the message concerning that selection and turns control over to the selected subroutine.

Figure 5-1 illustrates the DOS Menu selections and prompt message. The rest of this section describes each of these selections and gives examples of their uses.

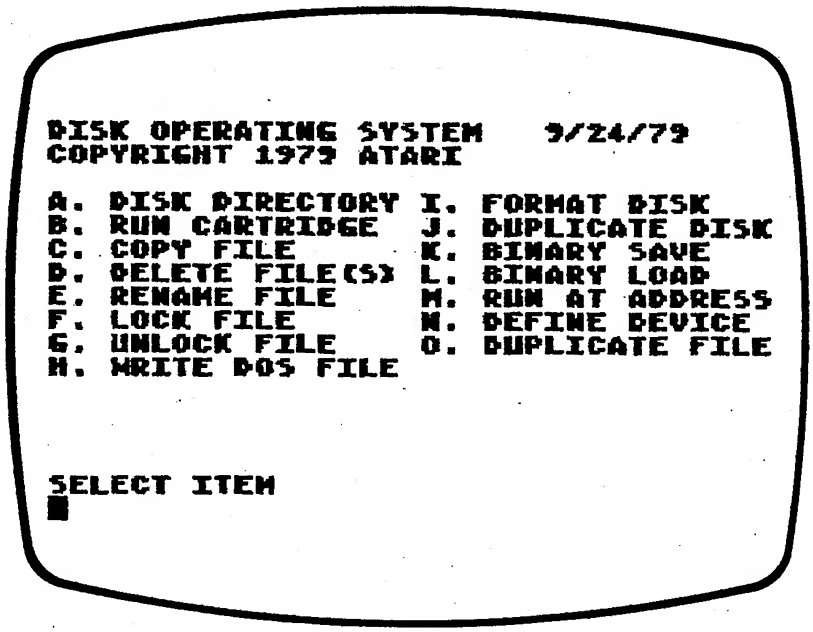

FIGURE 5-1. DOS MENU

A. DISK DIRECTORY

The disk directory contains a list of all the files on a diskette. On demand, it displays the filename, the extender (if any), and the number of sectors allocated to that file. It will either display a partial list or a complete list depending on the parameters entered. Wild cards can be used in the parameters.

Type A [RETURN] below the SELECT ITEM prompt.

The screen immediately displays the entry module message:

DIRECTORY--SEARCH SPEC,LIST FILE

You can choose at this time to search for a single filespec, several filespecs, or all filespecs on the diskette.

Type DOS.SYS [RETURN]

The screen displays:

```
    DOS     SYS 064
  645

     TYPE RETURN FOR MENU
```

This tells you that DOS.SYS is present on the diskette and contains 64 sectors. The number 645 on the next line notifies you of the number of free sectors left on the diskette. Since there are 709 sectors available to the user on a diskette, this display also lets the user know that DOS.SYS is the only file on the diskette.

The second parameter, LIST FILE, allows you to print the directory information to any output device you choose. Since there was no specified device in the example above, the DOS printed it on the screen (E:), which is the default device.

Notice that TYPE RETURN FOR MENU is displayed on the screen. Press the [RETURN] key and the DOS Menu redisplays with a SELECT ITEM prompt. To list the filespec names on a printer, type A [RETURN]. After the directory prompt message, type DOS.SYS,P: [RETURN]. If you have a printer and it is on, DOS will print DOS SYS 064 with 645 directly below it. If you do not have a printer (or it is not turned on), you will see an ERROR-138 displayed on the screen. Each time this selection completes a task, it displays a TYPE RETURN FOR MENU prompt message. The following examples illustrate different uses of this selection.

EXAMPLE 1:

```
   SELECT ITEM
   A [RETURN]
   DIRECTORY--SEARCH SPEC,LIST FILE?
   [RETURN]                              Lists all filenames on screen.


   TYPE RETURN FOR MENU [RETURN]
```

EXAMPLE 2:

```
   SELECT ITEM
   A [RETURN]
   DIRECTORY--SEARCH SPEC,LIST FILE?
   *.SYS [RETURN]                        Lists all files on screen with
                                         .SYS extender.


   TYPE RETURN FOR MENU [RETURN]
```

EXAMPLE 3:

```
SELECT ITEM
A [RETURN]
DIRECTORY--SEARCH SPEC,LIST FILE?
D2:,P: [RETURN]                    List all files on disk drive
                                   #2 on the line printer.

TYPE RETURN FOR MENU [RETURN]
```

EXAMPLE 4:

```
SELECT ITEM
A [RETURN]
DIRECTORY--SEARCH SPEC,LIST FILE?
DO?.* [RETURN]                     Lists all 3-letter filespecs
                                   beginning with DO.

TYPE RETURN FOR MENU [RETURN]

SELECT ITEM
```

B. RUN CARTRIDGE

Whenever you select B, the DUP file gives control of your ATARI Personal Computer System to the inserted cartridge. If the BASIC cartridge is inserted, the screen displays a READY prompt. If the Assembler Editor cartridge is inserted, the screen displays an EDIT prompt. If you have not inserted a cartridge, the message NO CARTRIDGE appears on the screen.

You can also press [SYSTEM RESET] to return control to the cartridge. This selection has no entry message or parameters.

EXAMPLE:

```
SELECT ITEM
B [RETURN]
```

C. COPY FILE

Use this selection if you have two disk drives and want to copy a file from one diskette in disk drive #1 to a second diskette in disk drive #2. You can also use this selection to create a backup copy on the same diskette with the same filename but a different extension, or even a completely different filename. No wild cards are allowed.

EXAMPLE 1:

```
SELECT ITEM
C [RETURN]
COPY--FROM,TO?
D1:DOSEX.BAS,D2:DOSEX.BAS [RETURN]  Copies DOSEX.BAS from D1 to D2
SELECT ITEM
```

EXAMPLE 2:

```
SELECT ITEM
C [RETURN]
COPY--FROM,TO?
D1:DOSEX.BAS,D1:DOSEX.BAK   Creates backup copy of file on
                            same diskette.
SELECT ITEM
```

Also, use this selection to copy the file listing to the screen, the printer, or the program recorder.

EXAMPLE 3:

```
C [RETURN]
COPY--FROM,TO?
D1:DOSEX.LST,E:  [RETURN]    Displays the program listing on
                             screen.

SELECT ITEM
```

EXAMPLE 4:

```
SELECT ITEM
C [RETURN]
COPY--FROM,TO?
E:,D1:TEMP.DAT [RETURN]      Copies any succeeding screen data
                             into a file named TEMP.DAT.
PETER [RETURN]               Type data on screen to
BILL [RETURN]                 be stored in TEMP.DAT file.
STEVE [RETURN]
[CTRL]3                       Terminates entry of data.
SELECT ITEM
```

EXAMPLE 5:

```
SELECT ITEM
C [RETURN]
COPY--FROM,TO?
D1:DISEX.LST,P:  [RETURN]    Lists the program listing DISEX.LST
                             on the printer.
SELECT ITEM
```

D. DELETE FILE(S)

This selection allows you to delete one or more files from the disk directory file and from the diskette. Wild cards can be used in the filespec names.

EXAMPLE 1:

```
SELECT ITEM
D [RETURN]
DELETE FILESPEC
D2:REM*.BAS [RETURN]          All files that begin with REM and that
                              have a .BAS extender.
TYPE "Y" TO DELETE...         Verification prompt.
REM1.BAS?                     Deletes REM1.BAS
Y [RETURN]
REMB.BAS                      Deletes REMB.BAS
Y [RETURN]
SELECT ITEM
```

EXAMPLE 2:

```
SELECT ITEM
D [RETURN]
DELETE FILESPEC
D:TEMP.DAT [RETURN]           A single file.
TYPE "Y" TO DELETE...         Verification prompt.
TEMP.DAT
Y [RETURN]                    If N is typed, file will not be
                              deleted.
SELECT ITEM
```

The verification prompt message gives you a chance to change your mind about deleting a file. By appending the option /N to the filespec entry, DOS will eliminate this verification step.

EXAMPLE 3:

```
SELECT ITEM
D [RETURN]
DELETE FILESPEC
DOXEX.BAS/N [RETURN]
SELECT ITEM                    File is deleted without requesting
                               verification.
```

You can also delete all files on a diskette, but leave the diskette formatting. Example 4 illustrates the steps for deleting all the existing files on disk drive #1. Note that the /N option is used in this example so that the verification message is not displayed.

EXAMPLE 4:

```
SELECT ITEM
D [RETURN]
DELETE FILESPEC
*.*/N [RETURN]                    Deletes all files from the disk
SELECT ITEM                       drive #1 diskette.
```

E. RENAME FILE(S)

This selection allows you to change the name of one or more files. You can use wild cards in the filespec names.

EXAMPLE 1:

```
SELECT ITEM
E [RETURN]
RENAME, GIVE OLD NAME, NEW
D2:TEMP.DAT,NAMES.DAT [RETURN]
SELECT ITEM                       The file designated on D2 as
                                  TEMP.DAT is changed to NAMES.DAT.
```

EXAMPLE 2:

```
SELECT ITEM
E [RETURN]
RENAME, GIVE OLD NAME, NEW
*.8KB,*.BAS [RETURN]              All files with extender 8KB have
SELECT ITEM                       their extenders changed to .BAS.
```

If you attempt to rename a file on a write-protected diskette, an ERROR-144 (Device Done Error) will display on the screen. If you try to rename a file that is not on the diskette, an ERROR-170 (File Not Found) error displays. If the screen displays ERROR-167, it means that you tried to rename a locked file (see F. LOCK FILE).

F. LOCK FILE

Use this selection to "write-protect" a single file. A locked file cannot be written to, appended, or deleted. An ERROR-167 will result from trying to write to a locked file. You can use wild cards to lock several files at the same time.

EXAMPLE 1:

```
SELECT ITEM
F [RETURN]
WHAT FILE TO LOCK?
DOS.SYS [RETURN]
SELECT ITEM
```

A locked file is designated by an asterisk (*) preceding its filename in the Disk Directory. Note that the asterisk has two different functions. When used within a filename, it is a wild card; when used preceding a filename in the DOS Directory, it denotes a locked file.

The following example illustrates the use of the wild card to lock all files with an extender of .BAS.

EXAMPLE 2:

```
SELECT ITEM
F [RETURN]
WHAT FILE TO LOCK?
D1:*.BAS [RETURN]
SELECT ITEM
```

EXAMPLE 3:

```
SELECT ITEM
F [RETURN]
WHAT FILE TO LOCK?
T*.* [RETURN]                    Locks all files on D1 which begin
SELECT ITEM                      with T.
```

EXAMPLE 4:

```
SELECT ITEM
F [RETURN]
WHAT FILE TO LOCK?
*.* [RETURN]                     Locks all D1 files.
SELECT ITEM
```

If you do not enter a filename or a wild card filename before pressing [RETURN], the screen will display an ERROR-165.

G. UNLOCK FILE

Use this selection to unlock a file or files that you previously locked using selection F. When you complete this selection, the asterisk that appeared before the filename in the DOS Directory that indicated that the file was locked will no longer appear on the screen. Wild cards can be used in the filespec names.

EXAMPLE 1:

```
SELECT ITEM
G [RETURN]
WHAT FILE TO UNLOCK?
DOSEX.BAS [RETURN]               Unlocks DOSEX.BAS file on D1.
SELECT ITEM
```

EXAMPLE 2:

```
SELECT ITEM
G [RETURN]
WHAT FILE TO UNLOCK?
T*.* [RETURN]                    Unlocks files beginning with
SELECT ITEM                      the letter T on disk drive #1
                                 diskette.
```

EXAMPLE 3:

```
SELECT ITEM
G [RETURN]
WHAT FILE TO UNLOCK?
PROB?.DAT [RETURN]               Unlocks all 5-letter files
SELECT ITEM                      beginning with PROB and having
                                 a .DAT extender.
```

H. WRITE DOS FILE

To write a DOS/FMS file on a diskette, the diskette must have been previously formatted (see I. FORMAT DISK). The diskette to be written is inserted in Disk Drive #1. This selection does not allow a choice of drives.

EXAMPLE:

```
SELECT ITEM
H [RETURN]
TYPE "Y" TO WRITE NEW DOS FILE?
Y [RETURN]
WRITING NEW DOS.SYS FILE
SELECT ITEM
```

You can also use C. COPY FILE or O. DUPLICATE FILE to write a new DOS onto a diskette, but you must first rename the DOS.SYS file before trying to copy it. After copying the file, rename it back to DOS.SYS and you will have a bootable version of DOS. However, since this is the selection designed specifically for writing a DOS/FMS file onto a second diskette, it is recommended that you use it rather than either of the two other selections, C or O.

I. FORMAT DISK

This selection is used to format a diskette. The diskette can be blank or it can have files on it that you do not want anymore. Formatting writes a digital pattern on the diskette that allows data to be stored and retrieved. Formatting a diskette takes approximately 2 minutes.

WARNING: Formatting a diskette always destroys all files existing on the diskette.

EXAMPLE:

```
SELECT ITEM
I [RETURN]
WHICH DRIVE TO FORMAT?
1 [RETURN]
TYPE "Y" TO FORMAT DISK 1
Y [RETURN]
SELECT ITEM
```

Although the above example illustrates disk drive #1 as the drive to be formatted, you can specify any drive. It is possible to format a diskette containing bad sectors. DOS will "flag" the bad sectors internally and will not write any data on those sectors.

J. DUPLICATE DISK

Use this menu selection to copy the files on a diskette (Source diskette) to another (Destination) diskette. The Disk Utility Package (DUP) program uses sector by sector copying, which means that programs already on the destination diskette will be destroyed.

This selection can be used with a one drive system or a multiple drive system. For a one drive system, save any RAM-resident BASIC program before attempting to duplicate a diskette. This selection uses the program area (where a RAM-resident BASIC program is stored) as a buffer for moving the files on the source diskette to the destination diskette when one drive is used.

EXAMPLE 1.

```
SELECT ITEM
J [RETURN]
DUP DISK--SOURCE,DEST DRIVES?
1,1 [RETURN]
TYPE "Y" IF OK TO USE PROGRAM AREA?
Y [RETURN]
INSERT SOURCE DISK, TYPE RETURN
[RETURN]
INSERT DESTINATION DISK, TYPE RETURN
[RETURN]
SELECT ITEM
```

Always write protect your source diskette as a safety measure. Then, if it is accidentally inserted when the destination diskette is supposed to be, the screen will display an ERROR-144, but your source diskette will still be intact.

If you type any character other than Y [RETURN] in response to the TYPE "Y" IF OK TO USE PROGRAM AREA message, the program aborts and a SELECT ITEM prompt appears on the screen.

NOTE: The number of times the DUP program requests you to insert the source and destination diskettes depends on the number and size of the file(s) to be duplicated for a given system and the amount of RAM in the system. A 48K system might require only two insertions whereas a 16K system might require five or six diskette insertions.

For a multiple disk drive system, it is not necessary to save a RAM-resident BASIC program, as an internal DOS buffer is used and the user's program area will not be altered. Notice that in the following example, there is no prompt message regarding the program area. Insert the source diskette in disk drive #1 and the destination diskette in disk drive #2 and use the steps in EXAMPLE 2.

EXAMPLE 2.

```
SELECT ITEM
J [RETURN]
DUP DISK--SOURCE,DEST DRIVES
1,2 [RETURN]
INSERT BOTH DISKS,TYPE RETURN
[RETURN]

SELECT ITEM
```

The cursor remains on the screen during the duplication process. This process can take several minutes if the source diskette is almost full.

Although the above example uses Disk Drive #1 as the source disk and #2 as the destination disk, you can duplicate a disk from any numbered drive to any other numbered drive; e.g., 2,1;3,1;2,3.

K. BINARY SAVE

Use this Menu selection to save the contents of memory locations in object file (binary) format. Programs written using the Assembler Editor Cartridge have this format. The parameters for this selection, START and END, are hexadecimal numbers. These numbers are determined by your object file program. For instance, if you had an Assembly Language program BINFIL.OBJ stored on diskette, it would be preceeded by a 6-byte header in the form shown in the following:

Header Byte #	Decimal Number	Hex Number	Description
#1	132	84	Identifier code for
#2	9	09	binary load file
#3	0	00	Starting address (LSB)
#4	60	3C	(MSB)
#5	255	FF	Ending address (LSB)
#6	91	5B	(MSB)
.	.	.	
.	.	.	Bytes of file data

In Example 1, you would insert the hexadecimal addresses 3C00 and 5BFF from the header as the START and END parameters in the BINARY SAVE selection.

EXAMPLE 1:

```
SELECT ITEM
K [RETURN]
SAVE--GIVE FILE,START,END
BINFIL.OBJ,3C00,5BFF [RETURN]
SELECT ITEM
```

This BINARY SAVE selection used with a /A after a filename can be used to incorporate an automatic run feature into an object file program. To make an object file run automatically, insert the starting address into hexadecimal locations 2E0 and 2E1. Since BASIC does not recognize hexadecimal numbers, use POKE statements with the decimal equivalents of the hexadecimal locations of the starting address. In the above example BINFIL.OBJ, the starting address is 3C00. The Least Significant Byte (LSB) is 00 and the Most Significant Byte is 3C. When these are converted to decimal the LSB is still 0, but the MSB is 60. The decimal equivalents of locations 2E0 and 2E1 are 736 and 737, respectively. See Appendix E.

EXAMPLE 2:

Binary File Automatic Run

Type B [RETURN] to get into BASIC

```
READY

POKE 736,00 [RETURN]                    Enter LSB first in decimal.
READY
POKE 737,60 [RETURN]                    Enter MSB second in decimal.
READY
DOS [RETURN]                            Return to DOS Menu.

SELECT ITEM
K [RETURN]
SAVE--GIVE FILE,START,END
BINFIL.OBJ/A,2E0,2E1 [RETURN]           Enter hexadecimal equivalents.
SELECT ITEM
```

When BINFIL.OBJ is loaded into RAM, it will begin executing immediately. Be sure to enter decimal numbers in the POKE statements. If you enter hexadecimal numbers by mistake, you could POKE into DOS and create all kinds of problems, making it necessary for you to reboot the system.

L. BINARY LOAD

Use this selection to load into RAM an Assembly Language (Binary) file that was previously saved with BINARY SAVE. If the starting address was appended to the file in locations 2E0 and 2E1, the file will automatically run after being entered.

EXAMPLE 1:

```
SELECT ITEM
L [RETURN]
LOAD FROM WHAT FILE?
BINFIL.OBJ [RETURN]
```

Since this file had the starting address in locations 2E0 and 2E1 (see K. BINARY SAVE), this file will begin executing as soon as the load is complete.

If a file has not had the starting locations inserted in 2E0 and 2E1, the SELECT ITEM prompt message would display on the screen as soon as the file completed loading.

EXAMPLE 2:

```
SELECT ITEM
L [RETURN]
LOAD FROM WHAT FILE?
MACHL.OBJ [RETURN]
SELECT ITEM
```

To run a program without an appended starting address, see the next selection, M. RUN AT ADDRESS.

M. RUN AT ADDRESS

Use this selection to enter the hexadecimal starting address of an object file program after you have loaded it into RAM with the BINARY LOAD selection. This selection is used when the starting address has not been appended to the object file.

EXAMPLE:

```
SELECT ITEM
M [RETURN]
RUN FROM WHAT ADDRESS?
3000 [RETURN]
```

In the above example, the instructions at hexadecimal location 3000 will begin executing. Be very careful in entering these hexadecimal address locations. If you enter an address that does not contain executable code, it will create problems. The least damaging of these problems causes the system to lock up making it necessary for you to reboot the system.

N. DEFINE DEVICE

This selection allows you to change the routing defined in the FMS Handler Table. For this use, it helps to think of E:, P:, and D: as single files rather than devices. In the following example, data that would have normally been sent to the printer is rerouted to a TEMP2.P file on a diskette.

EXAMPLE:

```
SELECT ITEM
N [RETURN]
LOGICAL DEVICE,PHYSICAL DEVICE
P:,TEMP2.P [RETURN]
SELECT ITEM
```

Within the system, the P: handler address is changed to reflect another handler in the FMS that "sets up" the actual device. In a program that has data to be sent to the printer, the FMS would find the address of P: in the Handler Table, and route the data to that location. However, this selection changes the P: "file" address and reroutes the data to the TEMP2.P file on the diskette. The logical device, which in this case is P:, can be redefined any number of times.

The full implementation of this selection is not supported, so use it with caution.

O. DUPLICATE FILE

Use this selection for a one drive system to copy a file from the diskette in drive #1 to another diskette. No wild cards are possible with this selection. The operation and prompt messages for this selection are very similar to those of the DUPLICATE DISK selection. The following example is for a single disk drive system.

EXAMPLE:

```
SELECT ITEM
O [RETURN]
NAME OF FILE TO MOVE?
DOSEX.BAS [RETURN]
TYPE "Y" IF OK TO USE PROGRAM AREA
Y [RETURN]
INSERT SOURCE DISK,TYPE RETURN
[RETURN]
INSERT DESTINATION DISK, TYPE RETURN
[RETURN]
SELECT ITEM
```

This selection also contains the same restrictions regarding the program area that are explained in Example 1 of the DUPLICATE DISK

selection. Therefore, you should not have a RAM-resident BASIC program that you do not wish destroyed when performing this task.

SECTION 6.

DISK OPERATIONS WITH BASIC

This section describes the BASIC commands that are used to move data between devices. Four of these commands allow storage and retrieval of files and the remainder are associated with input and output data operations. Each command is illustrated with its abbreviation, format, and an example of the command.

COMMANDS TO STORE AND RETRIEVE FILES

LOAD	LIST
SAVE	ENTER

LOAD (LO.)

Format: LOAD filespec
Example: LOAD "D1:DOSEX.BAS"

This command causes the computer system to load the filespec from the disk drive specified into RAM. It loads the tokenized version of the program. The tokenized is shorter than the untokenized version in that, when recorded, this version contains shorter inter-record gaps. However, when a tokenized version is loaded, it also loads the program's symbol table. If the program is altered or deletions are made, the symbol table is NOT changed. This means that all variables which were defined in the original program still exist in the symbol table. Therefore, it is recommended that LOAD and SAVE be used only when saving a program is its final form. When specifying disk drive #1, it is not necessary to call it out; e.g., D: is the same as D1:.

This command is also used in "chaining" programs. If you have a program that is too big to run in your available RAM, you can use the LOAD command as the last line of the first program so that when the program encounters the LOAD statement, it will automatically read in the next part of the program from the diskette. However, the second program must be able to stand alone without depending on any variables, etc. from the first program.

To cause the second segment to load and run automatically, you would use a RUN "D:filespec" as the last line of the first segment. However, before running the first segment, make sure you have saved it on diskette as the RUN statement will wipe out your RAM-resident program.

SAVE (S.)

Format: SAVE filespec
Example: SAVE "D1:EXAMP2.BAS"

This command causes the computer system to save a program on disk with the filespec name designated in the command. SAVE is the complement of

LOAD and stores programs in tokenized form.

LIST (L.)

Format: LIST filespec
Example: LIST "D:DATFIL.LST"

In BASIC, if no device is specified after the LIST command; e.g., LIST or L. 10,100, then the default device is the screen. All program lines currently RAM-resident or specified program lines are displayed on the screen. However, if a device is specified; e.g., P:, C:, D:, D2:, the RAM-resident program lines (or designated program lines) will be listed to the specified device.

When a disk drive is specified, this command causes the computer system to move the current RAM-resident (source) program to a file on diskette under the name specified by the referenced filespec. This command unlike SAVE, saves the untokenized (textual) format.

The untokenized format, although longer, does not merely enlarge on the already existing symbol table. Each time the program is changed and LISTED to the diskette, an updated symbol table is saved with it. This leads to less ERROR-9 and ERROR-5 messages as the variables are all current. When you are working with a program, it is advisable to save it using the LIST command.

ENTER (E.)

Format: ENTER filespec
Example: ENTER "D:LIST2.LST"

This command causes the computer system to move a file on diskette with the referenced filespec into RAM. The program is entered in untokenized form and is interpreted as the data is received. ENTER, unlike LOAD, will not destroy a RAM-resident BASIC program, but will merge the RAM-resident program and the disk file being loaded. If there are duplicate line numbers in the two programs, the line in the program being ENTERed will replace the same line in the RAM-resident program.

DISK INPUT/OUTPUT COMMANDS

An I/O operation is controlled by an I/O Control Block (IOCB). An IOCB is a specification of the I/O operation, consisting of the type of I/O, the buffer length, the buffer address and two more auxiliary control variables, of which the second is usually 0. ATARI BASIC sets up the eight IOCBs and dedicates three to the following:

IOCB #0 is used by BASIC for I/O to E:
IOCB #6 is used by BASIC for I/O to S:
IOCB #7 is used by BASIC for LPRINT, CLOAD, and SAVE commands.

IOCBs #1 through #5 can be used freely, but the dedicated IOCBs should be avoided unless a program does not make use of one of the dedicated uses mentioned above.

The I/O commands defined in this section are:

OPEN/CLOSE
INPUT/PRINT
PUT/GET
STATUS
XIO

The NOTE and POINT commands are not supported in this version of DOS.

OPEN (O.)

Format: OPEN #iocb,aexp1,aexp2,filespec
Example: 100 OPEN #2,8,0,"D1:ATARI800.BAS"

The OPEN statement links a specific IOCB to the appropriate device handler, initializes any CIO-related control variables (see Glossary), and passes any device-specific options to the device handler. The parameters in this statement are defined as follows:

#	Mandatory character entered by user.
iocb	A number between 0 and 7 that refers to a device or file.
aexp1	Code number that determines the type of operation to be performed.

Code 4 = input operation
6 = disk directory input operation
8 = output operation
9 = end-of-file append operation
Code 9 allows program input from screen editor without user pressing [RETURN].
12 = input and output operation

aexp2	Device-dependent auxiliary code. An 83 (ASCII S) in this position causes the ATARI 820 Printer to print sideways; otherwise it is always 0.
filespec	Specific file designation (see Section 1 for filespec definition).

In the example, OPEN #2,8,0,"D1:ATARI800.BAS", IOCB #2 is opened for output to a file on disk drive #1 designated as ATARI800.BAS. If there is no file by that name on disk drive #1, the DOS creates one. If a file by that name already exists, the OPEN statement destroys that file and creates a new one. If the IOCB has already been opened, the screen displays an ERROR-129 (File Already OPENed.)

CLOSE (CL.)

Format: CLOSE #iocb
Example: 300 CLOSE #2

The CLOSE command closes devices or files that had been previously opened for read/write operations. The number following the mandatory # must be the same as the iocb reference number used in the OPEN statement. (See Example 1.) If the iocb has already been opened to one device and an attempt is made to open the same IOCB to another device without first closing the IOCB, ERROR-129 displays on the screen. The same IOCB cannot be used for more than one device at a time.

EXAMPLE 1:

```
10 OPEN #1,8,0,"D:FIL.BAS
20 CLOSE #1
```

INPUT (I.)

Format: INPUT [#iocb {,;}] {avar svar} [, {avar svar}...]

Examples: 100 INPUT #2;X,Y
100 INPUT #2;N$

This command is used to request data (either numerical or string) from a specified device. When used without a #iocb, the data is assumed to be from the default device (E:).

```
5 REM **CREATE DATA FILE**
7 REM **OPEN WITH 8 CREATES DATA FILE**
10 OPEN #1,8,0,"D:WRITE.DAT"
20 DIM WRT$(60)
30 ? "ENTER A SENTENCE NOT MORE THAN 60
CHARACTERS."
35 INPUT WRT$
38 REM **WRITE DATA TO DISKETTE**
40 PRINT #1,WRT$
45 REM **CLOSE DATA FILE**
50 CLOSE #1
55 REM ** OPEN DATA FILE FOR READ**
58 REM **OPEN WITH 4 IS A READ ONLY**
60 OPEN #1,4,0,"D:WRITE.DAT"
65 REM ** READ DATA FROM DISKETTE**
70 INPUT #1,WRT$
75 REM ** PRINT DATA**
80 PRINT WRT$
85 REM ** CLOSE DATA FILE**
90 CLOSE #1
```

Figure 6-1. INPUT and PRINT Program Example

In Figure 6-1, line 50 reads the user input from the keyboard (default device). In line 80, the INPUT statement reads the contents of the string from the opened file.

PRINT (PR. or ?)

Format: PRINT {#iocb / [exp]} [{; / ,} [exp]...]

Examples:
```
100 PRINT #2;X,Y
100 PRINT #2;A$
```

This command writes an expression (whether string or arithmetic) to the opened device with the same IOCB reference number. See Figure 6-1. If no IOCB number is specified, the system writes the expression to the screen, which is the default device. If the information is directed to a device that is not open, ERROR-133 displays on the screen.

PUT (PU.)

Format: PUT #iocb,aexp
Example: 100 PUT #6,ASC("A")

The PUT command writes a single byte (value from 0-255) to the device specified by the IOCB reference number. In the following example program, the PUT command is used to write each number you type into an array dimensioned as A(50). You can enter up to 50 numbers, each of which should be less than 256. See GET command for second half of this program.

```
10 GRAPHICS 0:REM PUT/GET DEMO
20 DIM A(50),A$(10)
30 GRAPHICS 0:? " PUT AND GET TO DISK PR
OGRAM EXAMPLE":?
40 ? "Is this to be a READ or a WRITE?":
INPUT A$:?
50 IF A$="READ" THEN 160
60 IF A$<>"WRITE" THEN PRINT "?":GOTO 40

70 REM WRITE ROUTINE
80 OPEN #1,8,0,"D1:EXAMPL1.DAT"
90 ? "Enter a number less then 256":INPU
T X
95 REM **WRITE NUMBER TO FILE**
100 PUT #1,X
110 IF X=0 THEN CLOSE #1:GOTO 130
120 GOTO 90
130 GRAPHICS 0:? :? "Read data in file n
ow?":INPUT A$:?
```

Figure 6-2. Partial Program Example Using PUT

This command is used to create data files or append data to an

existing file.

GET (GE.)

Format: GET #iocb,avar
Example: 100 GET #2,X

This command reads a single byte from the device specified by the IOCB reference number into the specified variable. The following program is the second part of the program example listed in the PUT command description. It allows you to retrieve each byte stored by the PUT command.

```
140 IF A$="NO" THEN END
150 IF A$<>"YES" THEN 130
160 REM READ OUT ROUTINE
170 OPEN #2,4,0,"D1:EXAMPL1.DAT"
180 FOR E=1 TO 50
185 REM **READ NUMBER(S) FROM FILE**
190 GET #2,G:A(E)=G
200 IF G=0 THEN GOTO 230
210 PRINT "BYTE # ";E;"=";G
220 NEXT E
230 CLOSE #2
```

Figure 6-3. Partial Program Example Using GET

When you run this program, it will print the numbers entered from the keyboard together with the byte number in which it was stored.

STATUS (ST.)

Format: STATUS #iocb,avar
Example: 100 STATUS #1,ERROR

This command is used to store the status of the referenced drive in a specified variable. If no errors are encountered, the status is 1; otherwise it is one of the error codes found in Appendix B. The following program example incorporates the STATUS command to test whether or not the disk drive is ready; i.e., whether the drive door is closed. If the disk drive door is open, the disk drive power is not on, the diskette has not been inserted, or the diskette has been incorrectly inserted, the screen displays an ERROR-138 or ERROR-139 and takes the user out of the program. Using the TRAP command, the

user can remain in the program while the STATUS command allows the user to determine and correct the situation causing the error.

```
10 GRAPHICS 0:REM TRAP/STATUS DEMO
20 DIM A(50),A$(10),O$(1)
30 GRAPHICS 0:? " PUT AND GET TO DISK PR
OGRAM EXAMPLE":?
40 ? "Is this to be a READ or a WRITE?":
INPUT A$:?
50 IF A$="READ" THEN 160
60 IF A$<>"WRITE" THEN PRINT "?":GOTO 40

70 REM WRITE ROUTINE
80 TRAP 400:OPEN #1,8,0,"D1:EXAMPL1.DAT"

90 ? "Enter a number less then 256":INPU
T X
100 PUT #1,X
110 IF X=0 THEN CLOSE #1:GOTO 130
120 GOTO 90
130 GRAPHICS 0:? :? "Read data in file n
ow?":INPUT A$:?
140 IF A$="NO" THEN END
150 IF A$<>"YES" THEN 130
160 REM READ OUT ROUTINE
170 TRAP 400:OPEN #1,4,0,"D1:EXAMPL1.DAT
"
180 FOR E=1 TO 50
190 GET #1,G:A(E)=G
200 IF G=0 THEN GOTO 230
210 PRINT "BYTE # ";E;"=";G
220 NEXT E
230 CLOSE #1
240 END
400 TRAP 40000:STATUS #1,ST:IF ST<>138 A
ND ST<>139 THEN PRINT "HELP":? ST:GOTO 4
30
410 ? "The disk drive door must be close
d!"
420 ? "Type Y if you closed the disk dri
ve door.";:INPUT O$
430 CLOSE #1:GOTO 40
```

Figure 6-4. Program Listing for STATUS Command

To test this program and the STATUS command, type in the program and SAVE it on diskette. Open the door of the disk drive before RUNning the program. The STATUS command will cause the error number to be printed and will give you a prompt message to close the disk drive door. After you close the door, the program will try again to open the file. This time no error will occur and the program will not trap but give you a ? prompt for your first input.

XIO (X.)

Format: XIO cmdno,#iocb,aexp1,aexp2,filespec
Example: 100 XIO 3,#6,4,0,"D:TEST.BAS"

The XIO command is a general input/output statement used for special operations. It is used when you want to perform the functions that would otherwise be performed using the DOS Menu selections. These XIO commands are used to open a file, read or write a record or character, close a file, store status, reference a location in a file for reading or writing, or to rename, delete, lock, or unlock a file.

The cmdno is used to designate just which of the operations is to be performed.

cmdno	OPERATION	EXAMPLE
3	OPEN	Same as BASIC OPEN
5	GET Record	
7	GET Characters	Similar to BASIC statements
9	PUT Record	GET, INPUT, PUT, PRINT
11	PUT Characters	
12	CLOSE	Same as BASIC CLOSE
13	STATUS Request	Same as BASIC STATUS
32	RENAME	XIO 32,#1,0,0,"D:SECC,PRTS.BAS"
33	DELETE	XIO 33,#1,0,0,"D:TEMP.BAS"
35	LOCK FILE	XIO 35,#1,0,0,"D:ATARI.BAS"
36	UNLOCK FILE	XIO 36,#1,0,0,"D:DOSEX.BAS"

The following program allows you to create a file for each month of the year into which you can enter the names and birthdays of your family and friends. The program uses XIO statements to create the file for each month, to lock and unlock each file as needed by the program, and to close the file when you are through with it.

Line 20 defines the disk file D:BIRTHDAY as FILE$. Then in line 170, FILE$ is opened with an XIO statement for input. The XIO statement in line 390 unlocks the proper file. The XIO statement in line 400 creates the file and allows you to write to the file. The next XIO statement, in line 430, closes the file and the next line's XIO statement locks the file to prevent it from being accidentally overwritten or erased.

To run this program, enter a number from 1 to 12. The program will check to see whether or not there are any entries in the file.

If there are none, the screen will display the message NO BIRTHDAYS IN followed by whatever month you selected. If you have made entries, the screen will display the names of the people and their birthdays for that month. In either event, the screen will display the names and birthdays for the month you selected and the succeeding month (as a failsafe feature against your forgetting an important birthday that comes at the first part of the next month). When you do not wish to see another file or make another birthday entry, type NO to each prompt message and the program will terminate.

```
5 GRAPHICS 0
10 DIM A$(5),D$(15),FILE$(20),DATE$(20),
MON$(20),ERR$(20),NAME$(40)
20 FILE$="D:BIRTHDAY."
30 ERR$="ERROR IN MONTH #"
100 GRAPHICS 0:? :? :? "PLEASE TYPE MONT
H NUMBER (1-12)"
110 TRAP 100:INPUT MONTH
120 TSTEND=0
130 MONTH=INT(MONTH)
140 IF MONTH<1 OR MONTH>12 THEN ? ERR$:G
OTO 100
145 GOSUB 1000+MONTH
150 FILE$(12)=STR$(MONTH)
160 EOF=0
170 TRAP 700:XIO 3,#2,4,0,FILE$
180 TRAP 600:FOR I=0 TO 1 STEP 0
190 INPUT #2;NAME$
200 INPUT #2;DATE$
210 EOF=EOF+1
220 IF EOF=1 THEN ? "BIRTHDAYS IN ";MON$
;" ARE:":?
224 TEMP=LENK NAME$)
225 NAME$(TEMP+1)=" "
226 NAME$(30)=" "
227 NAME$(TEMP+2,30)=NAME$(TEMP+1)
230 ? NAME$,DATE$
240 NEXT I
299 REM %MAKE NEW ENTRY IN BIRTHDAYS%
300 ? :? :? "WOULD YOU LIKE TO MAKE A NE
W BIRTHDAY ENTRY":INPUT A$
305 IF A$="YES" THEN GOTO 320
310 IF A$="NO" THEN PRINT "WOULD YOU LIK
E TO CHECK ANOTHER MONTH'S FILE?":INPUT
A$:?
315 IF A$="YES" THEN GOTO 20
318 IF A$="NO" THEN END
320 ? "PLEASE TYPE PERSON'S NAME"
330 INPUT NAME$
340 ? "PLEASE TYPE PERSON'S BIRTHDAY (MM
-DD-YY)"
350 INPUT DATE$
360 MONTH=INT(VAL(DATE$))
370 IF MONTH<1 OR MONTH>12 THEN ? ERR$,D
ATE$:GOTO 300
380 FILE$(12)=STR$(MONTH)
```


Figure 6-5. Sample Program Using XIO Statements (Page 1 of 2)

```
390 TRAP 400:XIO 36,#3,0,0,FILE$:OPEN #2
,9,0,FILE$:GOTO 410
400 CLOSE #2:XIO 3,#2,8,0,FILE$
410 PRINT #2;NAME$
420 PRINT #2;DATE$
430 XIO 12,#2,0,0,FILE$
440 XIO 35,#2,0,0,FILE$
450 GOTO 300
600 CLOSE #2:IF EOF=0 THEN ? "NO BIRTHDA
YS IN ";MON$
610 MONTH=MONTH+1
620 IF MONTH>12 THEN MONTH=1
630 TSTEND=TSTEND+1
640 IF TSTEND=1 THEN GOTO 145
650 GOTO 300
700 EOF=0:GOTO 600
1001 MON$="JANUARY":RETURN
1002 MON$="FEBRUARY":RETURN
1003 MON$="MARCH":RETURN
1004 MON$="APRIL":RETURN
1005 MON$="MAY":RETURN
1006 MON$="JUNE":RETURN
1007 MON$="JULY":RETURN
1008 MON$="AUGUST":RETURN
1009 MON$="SEPTEMBER":RETURN
1010 MON$="OCTOBER":RETURN
1011 MON$="NOVEMBER":RETURN
1012 MON$="DECEMBER":RETURN
3000 FILE$(12,12+LEN(STR$(MONTH)))=STR$(
MONTH):? FILE$
```

Figure 6-5. Sample Program Using XIO Statements (Page 2 of 2)

APPENDIX A

ALPHABETICAL DIRECTORY
OF BASIC RESERVED WORDS USED WITH DISK OPERATIONS

Note: The period is mandatory after all abbreviated keywords.

RESERVED WORD:	ABBREVIATION	BRIEF SUMMARY OF BASIC STATEMENT
CLOSE	CL.	I/O statement used to close a disk file at the conclusion of I/O operations.
DOS	DO.	This command causes the menu to be displayed. The menu contains all DOS utility selections.
END		Stops program execution; closes files; turns off sounds. Program may be restarted using CONT. (Note: END may be used more than once in a program.)
ENTER	E.	I/O command used to retrieve a LISTed program in untokenized (source) form. If a program or lines are ENTERed when a program is resident in RAM, ENTER will merge the two programs.
GET	GE.	Used with disk operation to input a single byte of data into a specified variable from a specified device.
INPUT	I.	This command requests data from a specified device. The default device is E: (Screen Editor).
LIST	L.	This command outputs the untokenized version of a program to a specified device.
LOAD	LO.	I/O command used to retrieve a SAVEd program in tokenized form from a specified device.
OPEN	O.	Opens the specified file for input or output operations.
PRINT	PR. or ?	I/O command causes output from the computer to specified output device.
PUT	PU.	Causes output of a single byte of data from the computer to the specified device.

SAVE	S.	I/O statement used to record a tokenized version of a program in a specified file on a specified device.
STATUS	ST.	Calls status routine for specified device.
TRAP	T.	Directs execution to a specified line number in case of an INPUT error, allowing user to maintain control of program.
XIO	X.	General I/O statement used in a program to perform DOS menu selections and specified I/O commands.

APPENDIX B
BASIC ERROR MESSAGES

ERROR CODE NO.	ERROR CODE MESSAGE
2	Insufficient memory to store the statement or the new variable name or to DIM a new string variable.
3	Value Error: A value expected to be a positive integer is negative, a value expected to be within a specific range is not.
4	Too Many Variables: A maximum of 128 different variable names is allowed.
5	String Length Error: Attempted to store beyond the DIMensioned string length.
6	Out of Data Error: READ statement requires more data items than supplied by DATA statement(s).
7	Number greater than 32767: Value is not a positive integer or is greater than 32767.
8	Input Statement Error: Attempted to INPUT a non-numeric value into a numeric variable.
9	Array or String DIM Error: DIM size is greater than 32767 or an array/matrix reference is out of the range of the dimensioned size, or the array/matrix or string has been already DIMensioned, or a reference has been made to an undimensioned array or string.
10	Argument Stack Overflow: There are too many GOSUBs or too large an expression.
11	Floating Point Overflow/Underflow Error: Attempted to divide by zero or refer to a number larger than +1E98 or smaller than -1E99.
12	Line Not Found: A GOSUB, GOTO, or THEN referenced a non-existent line number.
13	No Matching FOR Statement: A NEXT was encountered without a previous FOR, or nested FOR/NEXT statements do not match properly. (Error is reported at the NEXT statement, not at FOR).
14	Line Too Long Error: The statement is too complex or too long for BASIC to handle.

15 GOSUB or FOR Line Deleted: A NEXT or RETURN statement was encountered and the corresponding FOR or GOSUB has been deleted since the last RUN.

16 RETURN Error: A RETURN was encountered without a matching GOSUB.

17 Garbage Error: Execution of "garbage" (bad RAM bits) was attempted. This error code may indicate a hardware problem, but may also be the result of faulty use of POKE. Try typing NEW or powering down, the re-enter the program wihout any POKE commands.

18 Invalid String Character: String does not start with a valid character, or string in VAL statement is not a numeric string.

19 LOAD Program Too Long: Insufficient memory remains to complete LOAD.

20 Device Number Larger than 7 or Equal to 0.

21 LOAD File Error: Attampted to LOAD a non-LOAD file.

Note: The following are INPUT/OUTPUT errors that result during the use of disk drives, printers, or other accessory devices. Further information is provided with the auxiliary hardware.

128 BREAK Abort: User hit [BREAK] key during I/O operation.

129 IOCB* already open. OPEN statement within a program loop or IOCB already in use for another device or file.

130 Nonexistent Device specified. Device is not turned on or not attached.

131 IOCB Write Only. Command to a write-only device (Printer).

132 Invalid Command: The command is invalid for this device.

133 Device or File not Open: No OPEN specified for the device.

134 Bad IOCB Number: Illegal device number.

135 IOCB Read Only Error: Command to a read-only device.

136 EOF: End of File read has been reached. (NOTE: This message can occur when using cassette files.)

*IOCB refers to Input/Output Control Block. The device number is the same as the IOCB number.

137	Truncated Record: Attempt to read a record longer than 256 characters.
138	Device Timeout. Device doesn't respond. Check connections between peripheral equipment and console.
139	Device NAK: Garbage on serial port or bad disk drive.
140	Serial Bus input framing error.
141	Cursor Out of Range for particular mode.
142	Serial Bus Data Frame Overrun.
143	Serial Bus Data Frame Checksum Error.
144	Device Done Error (invalid "done" byte): Attempt to write on a write-protected diskette.
145	Read After Write Compare Error (disk handler) or bad screen mode handler.
146	Function not Implemented in handler.
147	Insufficient RAM for operating selected graphics mode.
160	Drive Number Error.
161	Too Many OPEN Files (no sector buffer available).
162	Disk Full (no free sectors).
163	Unrecoverable System Data I/O Error.
164	File Number Mismatch: Links on disk are messed up.
165	File Name Error.
166	POINT Data Length Error.
167	File Locked.
168	Command Invalid (special operation code).
169	Directory Full (64 files).
170	File Not Found.
171	POINT Invalid.

APPENDIX C

HOW TO OBTAIN MORE USEABLE RAM

1. RELEASE UNNECESSARY DISK DRIVE BUFFERS

The DOS is able to control up to four disk drives. If you have fewer than four drives, you can use some of the RAM that the DOS sets aside for programming to control four drives.

By POKEing certain RAM locations, you can simplify the DOS so that it is not carrying the excess capability for controlling more disk drives than are in your system.

Here's the RAM-saving technique, step by step:

- First "boot up" using your Master Diskette with the BASIC cartridge in its slot. Now remove the diskette and insert a new unformatted diskette.

- Select DOS item B (RUN CARTRIDGE) to get into BASIC, then POKE memory location 1802 with the value shown in the table below. (Use Direct Mode).

REQUIRED POKES

NUMBER OF DRIVES	POKE THIS VALUE INTO 1802	PRINT FRE(0)	BYTES SAVED (OVER 4-DRIVE CONFIGURATION)
1	1	21006*	397
2	3	20867*	258
3	7	20739*	130

*Assumes 16K system.

- Now do a PRINT PEEK(1802) and check that the value returned is the same number that you POKEd into that location. If it isn't, repeat the POKE step and check again.

- Go to DOS and use Menu item I (FORMAT DISK) to put the formatting onto your blank diskette. Now use item H (WRITE NEW DOS) to place the DOS.SYS file (with the new value of location 1802), on your new diskette.

- With the DOS file now on the new diskette, power down the computer to clear the RAM. Power up (boot up) again with the new diskette installed.

- Now go to BASIC again and do a PRINT PEEK(1802),FRE(0) in direct mode. Location 1802 should show the new number you POKEd into it, and the free RAM should reflect a significant memory saving.

2. CHANGE THE NUMBER OF BUFFERS USED TO REFLECT THE MAXIMUM NUMBER OF FILES THAT NEED TO BE OPEN.

   This procedure is exactly the same as the previous example except that you now POKE memory location 1801 with the number of files you wish to allow to be open simultaneously. This could be as small as 1 if you are certain that you will not be executing a program that needs to open more than 1 file. The value of this parameter in the DOS as supplied by ATARI is 3, which means that 3 files can be open simultaneously.

3. ELIMINATE DOS AND FMS WHEN THEY ARE NO LONGER NEEDED.

When you power up in the proper sequence (disk drive then console), the DOS and FMS are loaded from the diskette into RAM. When you later run a BASIC program, you do not need parts of the DOS that have to do with dsiplaying the menu and responding to your selections. You can release the RAM reserved for these functions with the following BASIC program. You can still use all the DOS functions that are controllable with BASIC keywords. You release 5384 bytes of RAM.

```
 10 POKE 10,35
 20 POKE 11,242
 30 POKE 12,136
 40 POKE 13,7
 50 POKE 1804,48
 60 POKE 1805,18
 70 TRAP 90
 80 X=USR(1928)
 90 X=USR(40968)
100 PRINT FRE(0) :REM LINE 100 FOR TESTING
```

Future versions of DOS and FMS will accomplish this operation automatically.

APPENDIX E

TABLE FOR HEXADECIMAL TO DECIMAL CONVERSION
(up to 4 Hex Digits)

FOUR HEX DIGITS							
4		3		2		1	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
A	40960	A	2560	A	160	A	10
B	45056	B	2816	B	176	B	11
C	49152	C	3072	C	192	C	12
D	53248	D	3328	D	206	D	13
E	57344	E	3584	E	224	E	14
F	61440	F	3840	F	240	F	15

For example, to convert the hex number 1234 to decimal, add the entries from each of the 4 columns in the table. For 1, use the column number 4, and so on.

```
1234 hex. =      4096
               + 512
               +  48
               +   4
               -----
              4660 dec.
```

Other examples:

```
EEDD hex. =     57344
              + 3584
              +  208
              +   13
              ------
             61149 dec.

AB hex. =         160
                + 11
                ----
               171 dec.
```

APPENDIX D

ATARI 400 and ATARI 800 MEMORY MAP (PARTIAL)

ADDRESS DECIMAL	ADDRESS HEXADECIMAL	CONTENTS
65535	FFFF	OPERATING SYSTEM ROM
	E000	
10879	2A7F	DISK OPERATING SYSTEM (2A7F-700)
		DISK I/O BUFFERS (current DOS)
9856	2680	
9885	267F	DISK OPERATING SYSTEM RAM
4864	1300	(current DOS)
4863	12FF	FILE MANAGEMENT SYSTEM RAM
1792	700	(current DOS)
1791	6FF	FREE RAM
1536	600	
1151	47F	OPERATING SYSTEM RAM (47F-200)
1021	3FD	CASSETTE BUFFER
999	3E7	PRINTER BUFFER
960	3C0	
511	1FF	HARDWARE STACK
256	100	
255	FF	PAGE ZERO
		FLOATING POINT (used by BASIC)
212	D4	
211	D3	BASIC or CARTRIDGE PROGRAM
210	D2	
209	D1	FREE BASIC RAM
208	D0	
207	CF	FREE BASIC and ASSEMBLER RAM
203	CB	
202	CA	FREE ASSEMBLER RAM
176	B0	---------- BASIC ZERO PAGE
128	80	ASSEMBLER ZERO PAGE

APPENDIX F

AUTO.SYS USAGE

You can use AUTO.SYS in limited ways to perform a task that you always want done each time you power-up the system. The following example illustrates one use of the AUTO.SYS file to control the margins of your display area. Normally, the margins are set at 2 and 38, respectively. If you want them to be 0 and 39 respectively, you must POKE the new values into decimal locations 82 and 83.

EXAMPLE:

```
   POKE 82,0 [RETURN]        Sets the left margin to 0
READY                        Note margin changes immediately.
POKE 83,39 [RETURN]          Sets the right margin to 39
READY
DOS
```

Note that this example uses the intermediate mode to enter data into memory. The next step is to create the AUTO.SYS file which will automatically reload the data each time the computer is turned on. To create a load format file, you now use the DOS Menu selection, Binary Save, to save the contents of memory locations 82 and 83 in a file called AUTO.SYS. The BASIC command SAVE cannot be used to save any data which has been stored in this fashion. If you followed the above example, the DOS Menu should be displayed on the screen. (Figure 5-1 shows an illustration of the DOS Menu.) Under the SELECT ITEM prompt message, type K [RETURN]. The screen will display a prompt message SAVE--GIVE FILE,START,END. Type AUTO.SYS,52,53 [RETURN]. The 52 and 53 are the hexadecimal equivalents of the decimal locations 82 and 83. When the SELECT ITEM prompt message appears again, it indicates the AUTO.SYS file has been saved in load file format on the diskette. Now, whenever the DOS is loaded from this diskette, AUTO.SYS will be automatically read into RAM and the margins will be 0 and 39.

If you press the [SYSTEM RESET] key, the margins will default to their original default values of 2 and 38. [SYSTEM RESET] re-initializes the Operating System, but does not re-boot the DOS.

Access:	The method (or order) in which information is read from, or written to diskette
Address:	A location in memory, usually specified by a two-byte number in hexadecimal or decimal format.
Alphanumeric:	The capital letters A-Z and the numbers 0-9, and/or combinations of letters and numbers. Specifically excludes graphics symbols, punctuation marks, and other special characters.
Array:	A one dimensional set of elements that can be referenced one at a time or as a complete list by using the array variable name and one subscript. Thus the array B, element number 10 would be referred to as B(10). Note that string arrays are not supported by BASIC, and that all arrays must be DIMensioned. A matrix is a two dimensional array.
ATASCII:	The method of coding used to store text data. In ATASCII (which is a modified version of ASCII, the American Standard Code for information Interchange), each character and graphics symbol, as well as most of the control keys, has a number assigned to represent it. The number is a one-byte code (decimal 0-255).
AUTO.SYS:	Filename reserved by Disk Operating System.
Backup DOS Disk:	An exact copy of original master DOS/FMS diskette. Always keep backups of your DOS/FMS master diskette and of all important data diskettes.
Baud Rate:	Signalling speed or speed of information interchange in bits per second.
Binary Load:	Loading a binary machine-language object file into the computer memory.
Binary Save:	Saving a binary machine-language object file onto a disk drive or program recorder.
Bit:	Abbreviation of "binary digit". The smallest unit of information, represented by the value 0 or 1.
Boot:	This is the initialization program that "sets up" the computer when it is powered up. At conclusion of the Boot (or after "booting up"),

the computer is capable of loading and executing higher level programs.

Break: To interrupt execution of a program. Pressing the BREAK key causes a break in execution.

Buffer: A temporary storage area in RAM used to hold data for further processing, input/output operations, etc.

Byte: Eight bits.

CIO: Central Input/Output Subsystem. The part of the OS that handles input/output.

CLOSE: To terminate access to a disk file Before further access to the file, it must be opened again. See OPEN.

Data: Information of any kind.

Debug: To isolate and eliminate errors from a program.

Decimal: A number base system using the digits 0 through 9. Decimal numbers are stored in binary coded decimal format in the computer. See Bit, Hexadecimal, and Octal.

Default: A condition or value that exists or is caused to exist by the computer until it is told to do something else. For example, the computer defaults to GRAPHICS 0 until another graphics mode is entered.

Delimiter: A character that marks the start or finish of a data item but is not part of the data. For example, the quotation marks (") are used by most BASIC systems to delimit strings.

Destination: The device or address that receives data during an exchange of information (and especially an I/O exchange). See Source.

Directory: A listing of the files contained on a diskette.

Diskette: A small disk. A record/playback medium like tape, but made in the space of a flat disk and placed in an envelope for protection. The access time for a diskette is much faster than for tape.

DOS: Disk Operating System abbreviation. The software or programs which facilitates use of a disk drive system.

DOS.SYS:	Filename reserved by Disk Operating System.
Drive Specification or Drivespec:	Part of the filespec that tells the computer which disk drive to access. If this is omitted the computer will assume Drive #1.
Drive Number:	An integer from 1 to 4 that specifies the drive to be used.
End of File:	A marker that tells the computer that the end of a certain file on disk has been reached.
Entry Point:	The address where execution of a machine-language program or routine is to begin. Also called the transfer address.
File:	An organized collection of related data. A file is the largest grouping of information that can be addressed with a single name. For example, a BASIC program is stored on diskette as a particular file, and may be addressed by the statements SAVE or LOAD (among others).
Filename:	The alphanumeric characters used to identify a file. A total of 8 numbers and/or letters may be used, the first of which must be a letter.
Filename Extender or Extension:	From 0 to 3 additional characters used following a period (required if the extender is used) after the filename. For example, in the filename PHONLIST.BAS. the letters "BAS" comprises the extender.
Filespec:	Abbreviation for file specification. A sequence of characters which specifies a particular device and filename. Do not create two files with exactly the same filespec. If you do, and they are both stored on the same diskette, you will not be able to access the second file. And, as they both have the same filespec, the DOS functions of Delete File, Rename File, and Copy File will act on both files.
Format:	To organize a new or magnetically (bulk) erased diskette onto tracks and sectors. When formatted, each diskette contains 40 circular tracks, with 18 sectors per track. Each sector can store up to 128 bytes of data.
Hexadecimal or Hex:	Number base system using 16 alphanumeric characters: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,and F.
IOCB:	Input/Output Control Block. An arithmetic expression that evaluates to a number

between 1 and 7. The IOCB is used to refer to a device or file.

INPUT: To transfer data from outside the computer (say, from a diskette file) into RAM. OUTPUT is the opposite, and the two words are often used together to describe data transfer operations: Input/Output or just "I/O". Note that the reference point is always the computer. OUTPUT always means away from the computer, while INPUT means into the computer.

Least Significant Byte: The byte in the rightmost position in a number or a word.

Kilobyte or K: 1024 bytes of memory. Thus a 16K RAM capacity actually gives us 16*1024 or 16,384 bytes.

Machine Language: The instruction set for the particular microprocessor chip used, which in ATARI'S case is the 6502.

Most Significant Byte: The byte in the leftmost position in a number or a word.

Null String: A string consisting of no character whatever. For example, A$="" stores the null string as A$.

Object Code: Machine language derived from "source code", typically from Assembly Language.

Octal: The octal numbering system uses the digits 0 through 7. Address and byte values are sometimes given in octal form.

OPEN: To prepare a file for access by specifying whether an input or output operation will be conducted, and specifying the filespec.

Parameter: Variables in a command or function.

Peripheral: An I/O device.

Record: A block of data.

Sector: The smallest block of data that can be written to a disk file or read from one. Sectors can store up to 128 bytes. Also called a "physical record".

Source: The device or address that contains the data to be sent to a Destination. See Destination.

Source Code:	A series of instructions, written in language other than machine language, which requires translation in order to be executed.
String:	A sequence of letters, characters, stored in a string variable. The string variables name must end with a $.
Tokenizing:	The process of interpreting textual BASIC source code and converting it to the internal format used by the BASIC interpreter.
Track:	A circle on a diskette used for magnetic storage of data. Each track has 18 sectors, each with 128 byte storage capability. There are a total of 40 tracks on each diskette.
Variable:	A variable may be thought of as a box in which a value may be stored. Such values are typically numbers and strings.
Write-Protect:	A method of preventing the disk drive from writing on a diskette. ATARI diskettes are write-protected by covering a notch on the diskette cover with a small sticker.

A

B

C

D

E

F